Programmation Objet Avancée

Projet long

Cas 2 : enchères multi threads

Présenté par Hamza Karhat

Encadrant : Éric Dallaire

Mise en situation :

On veut développer un prototype de mise aux enchères avec des agents autonomes. Des acheteurs et des vendeurs doivent entreprendre des rondes de négociations pour tenter de réaliser un échange de produit (une transaction). À la fin de la séance d'enchères, on affiche la liste des transactions. Les contraintes applicable au projet sont les suivantes :

- Le système doit être objet
- Ne pas développer l'interface graphique, c'est une première itération
- L'objectif est d'avoir un système un système fonctionnel
- Gestion des montants des enchères (retour des fonds, budget maximum, etc)
- Le système s'arrête par lui-même après un certain temps, aucune entrée nécessaire pour l'exécution du programme
- Tout produit mis au enchère produit une transaction
- Pour le développement du prototype
- Limiter le nombre de stratégies
- Utiliser un seule type de produit avec un minimum d'attributs
- Utiliser une « factory » qui va s'occuper de charger les acheteurs et vendeurs
- Représenter les acheteurs par des threads demeurant sur le marché tant qu'il lui reste des fonds
- Représenter les vendeurs par des threads demeurant sur le marché tant que son article n'a pas été vendu

Description de la solution proposée

Dans le cadre de mon travail long en programmation objet avancée, j'ai développé un programme qui simule une vente aux enchères.

Commençons par la classe Item. Elle défini comme sont nom l'indique les objets en vente durant l'enchère. On y retrouve les attribut définissant son nom, le prix de départ de son enchère, son vendeur, la plus grosse offre faite par un acheteur ainsi que l'acheteur le plus offrant. Elle contient une méthode *bid(Buyer bidder, int Price)* qui permet de redéfinir certains de ces attributs lorsqu'une offre supérieure à la précédente est faite. Par ailleurs elle contient aussi un certain nombre d'accesseurs, notamment le getter static "getInstance()" permettant de récupérer tous les objets de cette classe. Un attribut Item est défini dans la classe Seller représentant les vendeurs.

Chacun des vendeurs et des acheteurs est représenté par un thread. Les classes Seller et Buyer contiennent toutes les méthodes basiques nécessaires à la vente aux enchères. Seller définit la méthode *Sell()*, et Buyer définit *Bid()*. Toutes deux héritent de la classe Thread ce qui leur permet de redéfinir la méthode *Run()*. Cette dernière doit contenir l'algorithme définissant le comportement d'un acheteur et d'un vendeur. J'ai donc défini Seller et Buyer comme classes abstraites et leurs sous classes (BuyerA, BuyerB, BuyerC, SellerA, SellerB) redéfinissent la méthode abstraite *Strategy()* qui est appelée dans la méthode *Run()* et est responsable de ce comportement. Ainsi, chaque sous classe représente une stratégie différente pour les acheteurs ainsi que les vendeurs. Buyer et Seller contiennent également un arraylist Instances contenant toutes les instances de ces classes et accessible via le getter static getInstance().

Chaque sous classe de Buyer et de Seller définit une stratégie différente. Pour les acheteurs voici les stratégies implémentées :

- Doubler l'enchère précédente
- Ne jamais enchérir en premier
- Enchérir minimalement

  et pour les vendeurs :
- Vendre après un certain nombre d'enchères
- Vendre après que l'enchère ait atteint un certain prix minimal

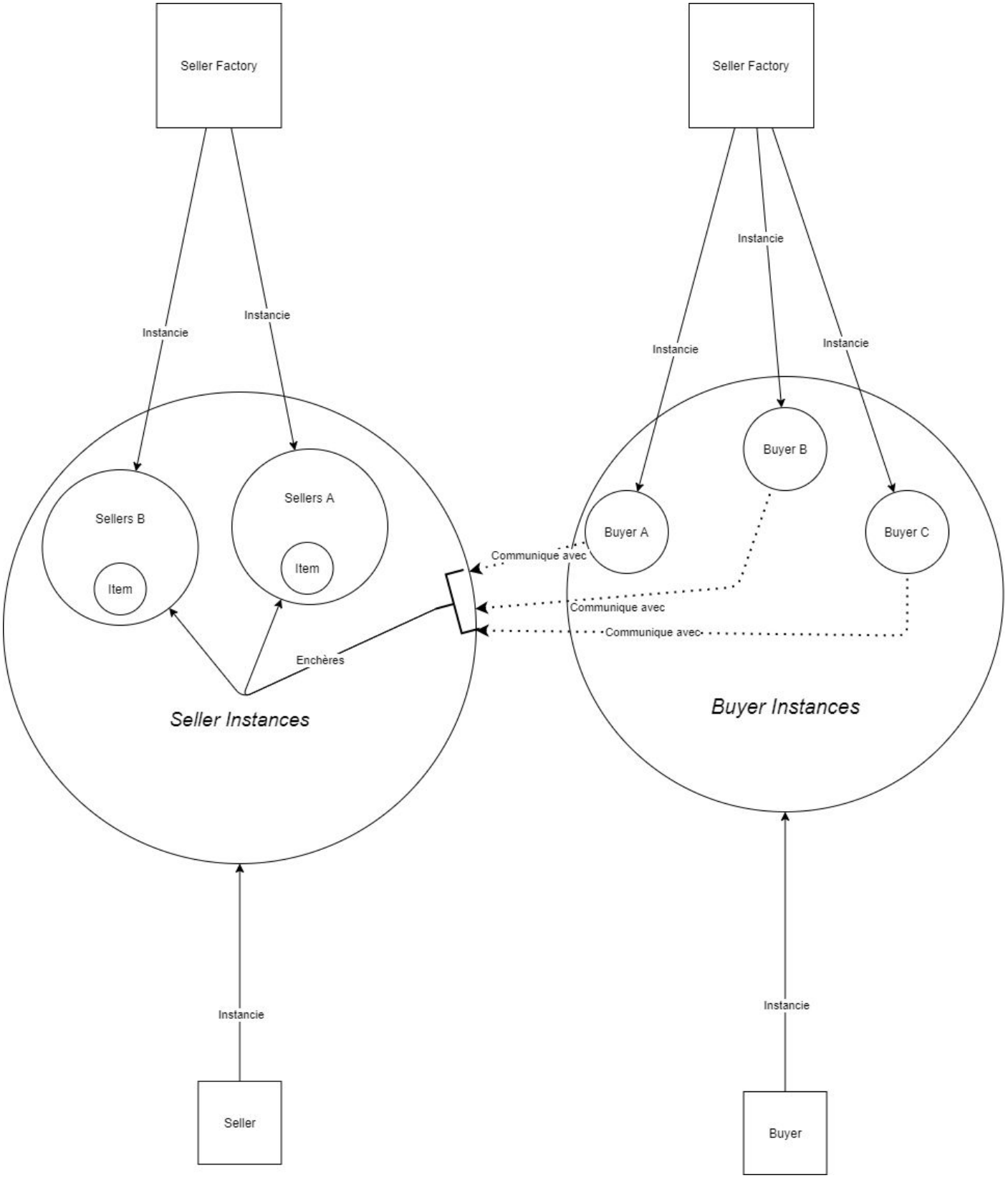
Buyer et Seller sont accompagnées de BuyerFactory et SellerFactory qui, grâce aux méthodes static *Build(int **x**)* qu'elles contiennent, créent un nombre ***x*** d'instances de Buyer ou Seller et les retourne sous forme d'un simple array. La répartition des stratégie sur les instances créées par les factory sont faites aux hasard.

La classe Auctioneer contient uniquement un constructeur qui s'occupe de lancer l'enchère en appelant les factory pour construire des instances de Buyer et Seller, et en utilisant leur méthode *Start()* qu'elles héritent de leur super classe Thread.

Le main contient simplement le constructeur *new Auctioneer*().

Points faibles et réflexions :

- Les Item sont instanciés par le constructeur de Seller avec des prix au hasard. Ce point faible sera normalement rectifié avant la remise de la version finale.
- Certains problèmes persistent : par moment tous les threads ne sont pas arrêtés malgré le fait que toutes leurs tâches ont été exécutées.
- Le projet n'est qu'un prototype. Il s'agit néanmoins d'une bonne base en vue de faire un projet complet simulant différentes stratégies d'enchères. Une version complète du projet aurait inclus une interface graphique, plus de stratégies voire un système de stratégies plus flexible permettant une implémentation rapide de nouvelles stratégies.
- Je pense que le projet pourrait bénéficier d'une meilleure gestion de la sécurité d'accès aux classes. Dans un cas réel, surtout si le modèle définissant les stratégies dans des sous classes est gardé, une meilleure gestion des packages est à considérer. Un package contenant les sous-classes définissant les stratégies serait par exemple envisageable.
- Je pense que le projet pourrait être amélioré en utilisant de la métaprogrammation : on pourrait créer une classe définissant stratégie personnalisée par l'utilisateur en lui demandant des paramètres (exemple le minimum d'enchères avant de sortir, ou si oui ou non il aimerait quitter l'enchère tôt si les acheteurs cessent d'enchérir). On pourrait essentiellement mettre en place une classe "Blueprint" qui créerait une stratégie personnalisée par quelques questions posées à l'utilisateur.


Seller Factory
Instancie
Instancie
Sellers B
Item
Sellers A
Item
Enchères
Seller Instances
Instancie
Seller
Seller Factory
Instancie
Instancie
Instancie
Buyer B
Buyer A
Buyer C
Communique avec
Communique avec
Communique avec
Buyer Instances
Instancie
Buyer

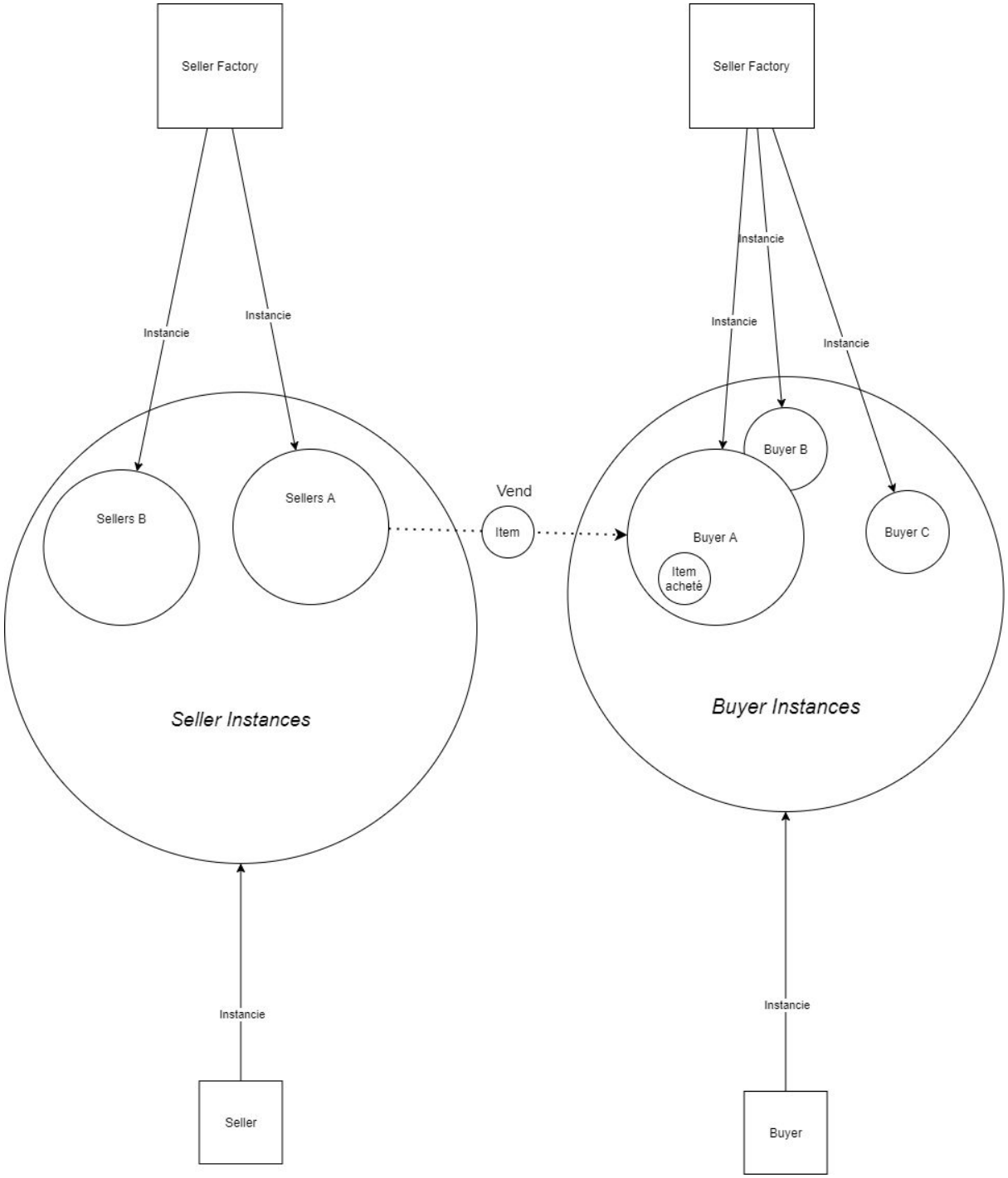
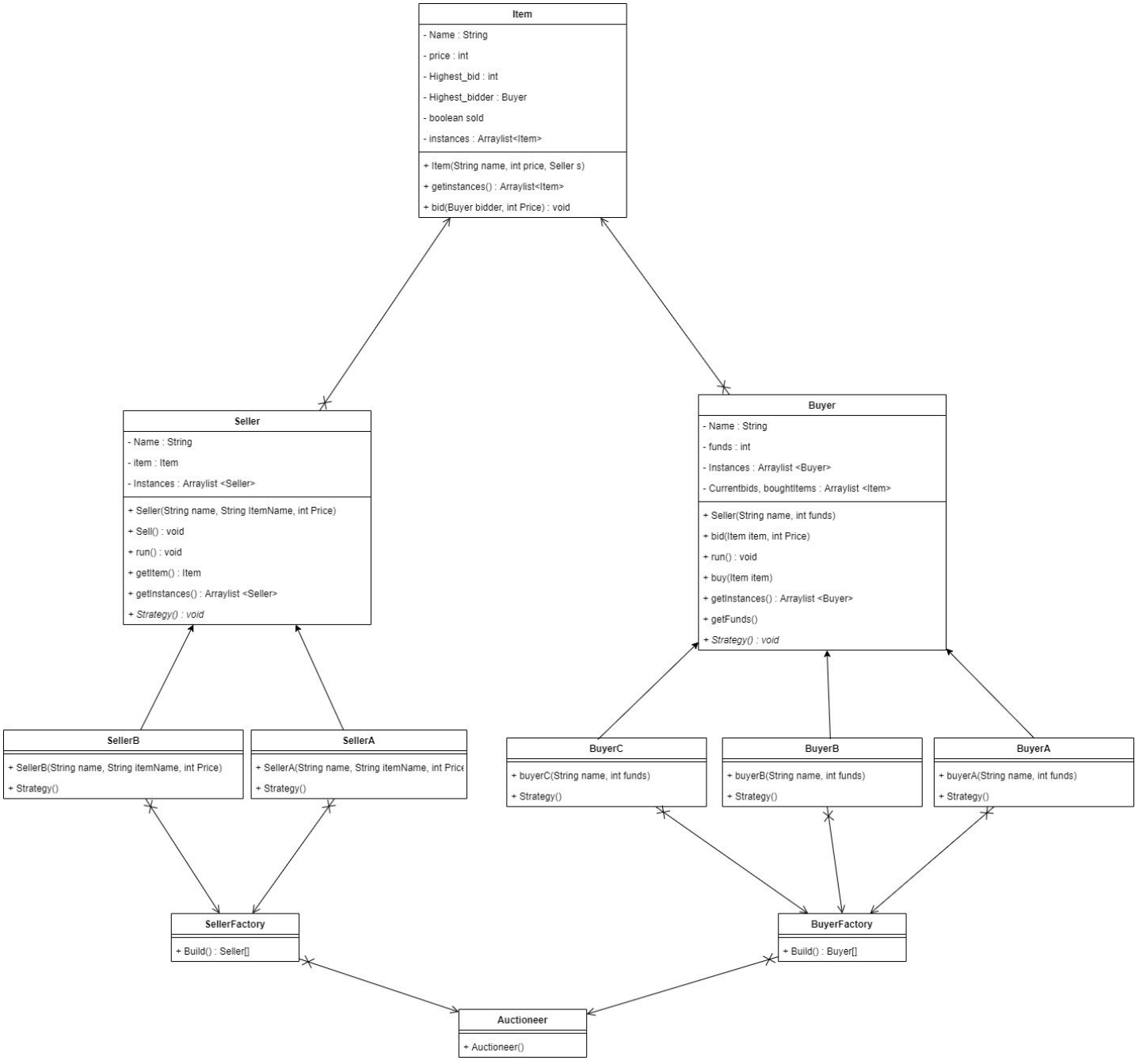

Seller Factory
Instancie
Instancie
Sellers B
Sellers A
Seller Instances
Vend
Item
Instancie
Seller
Seller Factory
Instancie
Instancie
Instancie
Buyer B
Buyer A
Item acheté
Buyer C
Buyer Instances
Instancie
Buyer

Diagramme de classe



Preuve de fonctionnement

```
"C:\Program Files\Java\jdk-11.0.2\b
Seller 3 running
Seller 1 running
Seller 5 running
Seller 4 running
Seller 2 running
buyer 5 starting with 12224
buyer 16 starting with 6460
buyer 12 starting with 7864
buyer 13 starting with 5001
buyer 14 starting with 6272
buyer 1 starting with 6204
buyer 9 starting with 12562
buyer 8 starting with 10007
buyer 10 starting with 8299
buyer 4 starting with 13307
buyer 7 starting with 10111
buyer 11 starting with 14500
buyer 6 starting with 7553
buyer 3 starting with 14007
buyer 15 starting with 8015
buyer 2 starting with 5600
buyer 5 bids on Item 3 for 458!
buyer 2 bids on Item 2 for 674!
buyer 15 bids on Item 5 for 226!
buyer 3 bids on Item 5 for 452!
buyer 6 bids on Item 2 for 1348!
buyer 11 bids on Item 4 for 254!
buyer 7 bids on Item 4 for 508!
buyer 12 bids on Item 4 for 1016!
buyer 12 bids on Item 4 for 1016!
buyer 4 bids on Item 3 for 916!
buyer 10 bids on Item 1 for 734!
buyer 16 bids on Item 5 for 456!
buyer 8 bids on Item 3 for 1832!
buyer 9 bids on Item 2 for 2696!
buyer 1 bids on Item 5 for 912!
buyer 14 bids on Item 5 for 1824!
buyer 13 bids on Item 1 for 1468!
buyer 1 bids on Item 5 for 3648!
buyer 16 bids on Item 5 for 3684!
buyer 10 bids on Item 1 for 2936!
buyer 4 bids on Item 3 for 3664!
buyer 7 bids on Item 4 for 2032!
buyer 11 bids on Item 4 for 4064!
buyer 6 bids on Item 2 for 5392!
buyer 3 bids on Item 5 for 7368!
buyer 15 bids on Item 5 for 8015!
buyer 2 bids on Item 2 for 5600!
buyer 5 bids on Item 3 for 7328!
buyer 3 bids on Item 5 for 14007!
buyer 6 bids on Item 2 for 7553!
buyer 7 bids on Item 4 for 8128!
buyer 4 bids on Item 3 for 13307!
buyer 9 bids on Item 2 for 12562!
buyer 13 bids on Item 1 for 5001!
buyer 10 bids on Item 1 for 8299!
buyer 16 exiting..
buyer 11 bids on Item 4 for 14500!
buyer 12 exiting...
```

```
buyer 12 exiting..
buyer 15 exiting...
buyer 15 exiting..
buyer 2 exiting...
buyer 2 exiting..
buyer 6 exiting...
buyer 6 exiting..
buyer 1 exiting...
buyer 1 exiting..
buyer 13 exiting...
buyer 13 exiting..
buyer 14 exiting...
buyer 14 exiting..
Seller 3 exiting...
Seller 4 exiting...
Seller 2 exiting...
Seller 5 exiting...
buyer 10 exiting...
buyer 10 exiting..
Item 1 was sold to buyer 10
Seller 1 exiting...
buyer 8 exiting...
buyer 8 exiting..
buyer 3 exiting...
buyer 3 exiting..
buyer 9 exiting...
buyer 9 exiting..
buyer 11 exiting...
buyer 11 exiting..
buyer 5 exiting...
```

```
Seller 5 exiting...
buyer 10 exiting...
buyer 10 exiting..
Item 1 was sold to buyer 10
Seller 1 exiting...
buyer 8 exiting...
buyer 8 exiting..
buyer 3 exiting...
buyer 3 exiting..
buyer 9 exiting...
buyer 9 exiting..
buyer 11 exiting...
buyer 11 exiting..
buyer 5 exiting...
buyer 5 exiting..
buyer 4 exiting...
buyer 4 exiting..
buyer 7 exiting...
buyer 7 exiting..

Process finished with exit code 0
```